APPLICATION FOR UNITED STATES

LETTERS PATENT

SYSTEM AND METHOD FOR ACHIEVING STRONG DATA CONSISTENCY

Inventors:

> ARUN KWANGIL IYENGAR
>
> RICHARD P. KING
>
> GABRIEL GARCIA MONTERO
>
> DANIELA ROSU
>
> KAREN WITTING

SYSTEM AND METHOD FOR ACHIEVING

STRONG DATA CONSISTENCY

BACKGROUND OF THE INVENTION

5

1. Field of the Invention

The present invention relates to data storage and more particularly to systems and methods for achieving data consistency among multiple copies.

10

2. Description of the Related Art

Many computer applications create multiple copies of the same data. Maintaining consistency of these multiple copies is critically important. How the updating of the different copies is coordinated leads to different levels of consistency among the copies, in return for different costs to perform that coordination. Typically, a stronger consistency, with closer coordination between peer cache updates, results in a larger consumption of resources and larger worst-case completion time.

A problem of keeping multiple caches consistent with each other is evident in processor caches for

multiprocessors and file caches for distributed file

systems. For processor caches, response times must be

extremely fast (orders of magnitude faster than those for

Web caches). To achieve these high speeds, the caches have

5   extremely short and fast links of guaranteed reliability to

a memory controller that permits them to be informed

simultaneously of updates. Techniques that work well given

those facilities are simply not practical for distributed

applications such as Web caches.

10   The Andrew File System (AFS) uses a weak consistency

method, where the server informs clients of updates. This

weak consistency scheme, with the clients checking with the

server (see e.g., J. Howard, M. Kazar, S. Menees, D.

Nichols, M. Satyanarayanan, R. Sidebotham, and M. West in

15   "Scale and performance in a distributed file system", ACM

Transactions on Computer Systems, 6(1):51-81, February

1988), can have significant overhead.

Therefore, a need exists for new consistency methods

which provide a high level of consistency guarantees

20   without the high overhead normally associated with such

methods.

SUMMARY OF THE INVENTION

A system and method for maintaining objects in storage elements includes maintaining information regarding which storage elements are storing particular objects and responding to a request to update an object by using maintained information to determine which of the storage elements store a copy of the object. Each storage element is instructed to invalidate the copy of the object, and an update of the object is performed after each storage element that includes the copy of the object indicates that the storage element has invalidated the copy of the object or the storage element is determined to be unresponsive.

In a system comprised of a plurality of storage elements, a method for maintaining stored objects includes maintaining a consistency coordinator which communicates with the storage elements and stores information regarding which storage elements are storing which objects. In response to receiving a request to update an object, information from the consistency coordinator is used to determine a set of storage elements which may store a copy of the object. Each storage element in the set is instructed to invalidate a copy of the object, and the

update is performed after each storage element in the set indicates that the storage element has invalidated a copy of the object or the storage element is determined to be unresponsive.

5   These and other objects, features and advantages of the present invention will become apparent from the following detailed description of illustrative embodiments thereof, which is to be read in connection with the accompanying drawings.

10

BRIEF DESCRIPTION OF DRAWINGS

The invention will be described in detail in the following description of preferred embodiments with reference to the following figures wherein:

15   FIG. 1 is a block/flow diagram of a system showing features of the present invention; and

FIG. 2 is a block/flow diagram showing a method for maintaining consistency between copies in accordance with the present invention.

20

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention discloses systems and methods for achieving data consistency among multiple copies. Several applications can make use of the present data

5   consistency methods including but not limited to storage elements, which may include caches, Web applications, file systems, memory storage devices and databases.

One distinction between the environment of a distributed file system and a Web environment, which makes

10   the present invention particularly useful, includes that in a Web environment, there is often only one source for changes for an object. Furthermore, in a Web environment, the types of object updates, e.g., one or multiple writers, is often known at the time of object creation.

15   The present invention will be illustratively described in terms of a cache consistency system and method; however, while the present invention is described in the context of caches, it should be clear to one of ordinary skill in the art that these techniques can be applied to application

20   states for a broad range of applications in addition to caches. It is also to be understood that objects as referred to herein may include any form of data, data sets, data blocks, and/or objects used in object-oriented

programming. The present invention integrates several cache consistency methods in a unique framework that enables the content-providing application to customize, on a per-object basis, the dissemination of cache updates to remote caches. For instance, in deployments with relatively large variations of transfer times between content provider and remote caches, the application can choose to use strong consistency methods only for a small subset of the objects, and weak consistency methods for the rest of the objects.

One feature of the present architecture for cache consistency includes a consistency coordinator. This coordinator can, among other things, manage transactions between a source(s) of object changes, the content provider(s), and the caches. Depending on which consistency model is being used for an object, the coordinator can take different actions.

The consistency methods provided by the consistency coordinator attempt to minimize the amount of network resources and worst-case completion times. For instance, the coordinator may keep track of which caches store which objects and restrict the update notification procedure to just those caches that have the object.

Cache consistency methods

When multiple copies of an object exist within a system, a key problem is how to ensure that, upon object updates, clients reading the various copies obtain

5   "consistent" content. The semantics of "consistent" depends on, e.g., system requirements. At one end, the system can provide strong consistency, ensuring that at any time, a request to read an object is satisfied with the latest version of the object. At the other end, the system

10   can provide weak consistency, ensuring that a read returns a value for the object, which was current at some point in the past.

Strong consistency may need a tight coordination of updates of copies of an object. In a system of peer caches,

15   one has to ensure that at the time when a new version of an object becomes available, no peer cache can serve an earlier version. Therefore, all the cached copies of an object should be invalidated before an update takes place in any of the caches.

20   Weak consistency does not require the coordination of updates; individual caches can acquire and serve the latest version of an object even if peer caches have not invalidated their old versions. Therefore, weak

consistency methods do not guarantee that all caches

storing a copy of the object will receive messages and

process them at exactly the same time. Namely, during an

object update, in the time interval between the first and

5   the last cache receiving their invalidation messages, a

client that requests for the updated object, which reaches

different caches, can receive different versions of the

object. The likelihood of this inconsistency increases

when there is a wider variance in communication times

10   between the individual caches and the content

provider/coordinator.

Weak consistency methods can differ in how long a time

it takes and how many system resources are consumed for

updating all object copies with the latest version. In

15   comparison to weak consistency methods, strong consistency

methods are likely to need more message exchanges and may

result in a longer time interval in which the object is not

accessible. The difference becomes relevant when the

distance between content provider and peer caches

20   increases.

It should be understood that the elements shown in

FIGS. may be implemented in various forms of hardware,

software or combinations thereof. Preferably, these

elements are implemented in software on one or more

appropriately programmed general-purpose digital computers

having a processor and memory and input/output interfaces.

Referring now to the drawings in which like numerals

5   represent the same or similar elements and initially to

FIG. 1, a system 10 having a plurality of caches 13 storing

data from one or more content providers 11 is

illustratively shown. In one scenario, one or more writers

perform updates to cached data. It is possible for a

10   writer 14 and a cache 13 to reside on the same node or to

constitute the same entity. Similarly, it is possible for

a writer 14 and a content provider 11 to reside on the same

node or to constitute the same entity.

The consistency coordinator 12 coordinates

15   interactions among content providers 11, writers 14, and

caches 13. Consistency coordinator 12 may be distributed

across multiple nodes and/or multiple consistency

coordinators 12 may exist in the system. The use of

multiple consistency coordinators can result in higher

20   availability, as the system may be able to function in the

event of a failure of less than all of the consistency

coordinators. Multiple consistency coordinators can also

increase the throughput of the system and thus improve

performance.

Although content provider 11, writer 14, caches 13, are depicted in FIG. 1 with communication paths to consistency coordinator 12, it is possible to have other communication paths in the system within the spirit and scope of the invention. As one such example, a writer 14 may communicate with a content provider 11 directly. Communication may also be achieved by employing heartbeat messages 27 as will be explained below.

Weak Consistency

For weak consistency paths, expiration-time consistency will now be addressed. Expiration-time consistency is a method used for Web caches, which communicate with content providers via HTTP. The content provider assigns to each object an expiration time. Consistency is managed by caches obeying expiration times. Namely, if an object is requested after its expiration time, the cache contacts the content provider to obtain the latest version of the object or, if the object has not changed, the new expiration time.

Update-all consistency addresses the problem of single-writer updates. With this method, consistency is

managed by sending consistency messages to all caches

whenever an object changes. The type of consistency

message depends on the implementation and object

characteristics. Generally the message instructs the cache

5   to invalidate any local version of the identified object it

may have.

Caches send an acknowledgment that they have received and

successfully processed the invalidation message. If they

fail to respond within a timeout period, the message is

10   resent. If a cache fails to respond after several retries,

special action is taken.

Update-holders consistency addresses the problem of

single-writer updates. This method is similar to update-

all consistency except that consistency messages are only

15   sent to caches that are storing the object. The

consistency coordinator maintains information that

indicates which caches are storing which objects. This

information is used when an object update occurs to create

the list of caches to which invalidation messages are to be

20   sent. To enable this ability, the consistency coordinator

may act as a reverse proxy between the content provider and

the caches. In some cases, a consistency coordinator may

not have exact information about which caches are storing

which objects.   In these situations, the consistency

coordinator can still use the information that it has to

make intelligent choices.

When an object needs to be updated, the coordinator

determines which caches include the object and sends

consistency messages only to those caches.   To maintain an

accurate list of which caches include which objects the

coordinator updates its state when the following types of

operations occur:

1. when a cache miss is served.   The cache sends a GET

request to the consistency coordinator, which will

update its state appropriately.

2. when a cache discards an object.   The cache notifies

the consistency coordinator that the object is no

longer in the cache.

3. when an object is updated.   The coordinator manages

the sending of invalidation messages and updates its

state appropriately.

The consistency coordinator may be a single entity or

may run across multiple applications and/or nodes.   If a

consistency coordinator is running on multiple nodes, one

method for achieving high availability and high throughputs

is for each consistency coordinator node to maintain

information about different sets of objects. Based on the name of the object, the consistency coordinator node corresponding to the object could be determined. There are several methods for assigning objects to consistency

5   coordinator nodes including hashing based on the object name.

Assigning objects to consistency coordinator nodes should be done in a manner which distributes load evenly across the consistency coordinator nodes. If one node of a

10 consistency coordinator fails, then the system only loses information about where objects are stored for the objects corresponding to the failed node, not all of the objects. It is also possible to have redundancy in how objects are assigned to consistency coordinator nodes. That way, the

15 caches storing an object could be determined from more than one consistency coordinator. This adds additional fault tolerance since even less information may be lost in the event of a cache failure.

Update-Local-Copy consistency addresses the problem of

20 multiple-writer updates. With this method, a writer accesses its local copy, performs the updates, and sends the new content to the consistency coordinator. The coordinator pushes the content to other caches using either

update-all or update-readers consistency methods.
Optionally, the coordinator sends an acknowledgement of the
update to the writer.

If the updated content arrives while the coordinator
is in the process of pushing another update for the same
object, it will save the newly arrived content until the
current update procedure is completed. If another version
of the object is already waiting for update, this version
is discarded and the newly received version is saved.

Update-Global-Copy consistency addresses the problem
of multiple-writer updates. Different than Update-Local-
Copy, in this method, the writer updates the most recent
version existing in the system.

Towards this end, before the update, the writer
contacts the consistency coordinator to retrieve the most
recent version of the object. The consistency coordinator
sends the content, or acknowledges that the local copy in
the write cache is the most recent. Upon sending the
reply, the coordinator records a write lock for the object
held by the writer and assigns it a lock timeout.

Upon receiving the most recent version of the object,
the writer performs the update and sends the new version to
the consistency coordinator, which cancels the write lock,

and distributes the new content to the other caches using

either update-all or update-readers consistency methods.

Optionally, the coordinator sends an acknowledgement of

update to the writer cache.

5   If the consistency coordinator receives another

request for update before the current write lock for the

object is either released or expires, it postpones the

reply until the update is received or the write lock

expires. In the former case, the new version is sent to the

10   requesting node and a new write lock is set for the object.

In the latter case, the writer cache is sent a negative

acknowledgment of update, and the coordinator sends the

available version of the object to the requesting node and

a new lock is set for the object. Upon receiving a

15   negative acknowledgement, the cache invalidates the updated

version, if already created, and may reinitiate the update

procedure. If an update completes before the previous

version was fully distributed to caches (according to the

chosen protocol), the coordinator saves the new content and

20   acts as indicated for update-local-copy if the second

update completes before the distribution completes. Read

requests which arrive at the coordinator for an object with

a write lock are responded to with the most recent version

available on the coordinator.

The expiration-time consistency method is limited by
the ability of the content provider to provide a good
5   estimate for when an object is to expire. In many
circumstances, this is not possible, and an object is
updated before its expiration time. If only HTTP is used
to communicate between content provider and caches, when
the update occurs, the content provider has no way of
10   initiating object invalidation or expiration-time change,
thus the cache continues to serve the obsolete version.

Update-all and Update-holders consistency methods do
not exhibit this limitation. By sending messages that
invalidate an updated object or that simply change its
15   expiration time to the time of the actual update, these
methods can provide better consistency than expiration-time
consistency. Comparing Update-holders and Update-all
methods, the former method needs fewer consistency messages
if many of the updated objects are not present in all
20   caches. This benefit is more relevant when the update rate
is relatively high.

However, Update-holders has the disadvantage that the
consistency coordinator has to be notified of any cache

update. If caches are modified frequently, the coordinator could become a bottleneck. A more scalable solution is to have the caches batch discard notifications, instead of sending them as they occur; this approach diminishes the

5   difference in consistency messages between Update-holders and Update-all methods.

Strong Consistency Methods

Coordinate-all consistency is based on the idea that upon an update, caches invalidate their copy of the updated

10   object before any of the caches can serve the new version of the object. More specifically, upon an object update, before making the new version available, the consistency coordinator sends invalidation messages to remote caches. A cache invalidates its copy of the object, if available,

15   and acknowledges the invalidation request.

The consistency coordinator waits to receive acknowledgments from caches. If a cache fails to respond within a timeout period, the invalidation message is resent, up to a preset limit on the number or duration of

20   retries. If this limit is reached, the cache is declared inaccessible and an implementation specific mechanism ensures that if active, the cache stops serving objects.

Once caches have acknowledged the notification or have

been declared inaccessible, the consistency coordinator

allows access to the new version of the object. Requests

for the updated object that arrive at a cache after the

invalidation message has been processed are handled in the

5   way of a traditional cache miss, meaning that the cache

sends a request to the coordinator for the first request

and waits for a reply, queuing subsequent requests behind

the first one. The coordinator reply depends on the stage

of the consistency procedure.

10   Coordinate-holders consistency addresses the problem

of single-writer updates. The method is based on the idea

that an object update procedure like the one defined for

Coordinate-all consistency should only involve those caches

that will access the object without validation.

15   Coordinate-holders consistency is similar to update-holders

in that the consistency coordinator maintains information

that indicates which caches are storing which objects.

When the writer/content provider wishes to update an object

it contacts the consistency coordinator. The coordinator

20   notifies caches currently storing the object to invalidate

their copy of the object. When these caches have

acknowledged the request, the coordinator makes the new

version of the object available.

If a cache fails to acknowledge the invalidation message the coordinator retries the request until it receives a response, up to a preset limit on the number or duration of retries. If this limit is reached, the cache is declared inaccessible and an implementation specific mechanism ensures that if active, the cache stops serving objects.

Referring to FIG. 2 with continued reference to FIG. 1, a method for achieving strong consistency in accordance with the present invention is depicted. Block 21 is constantly active as the system executes. The consistency coordinator 12 maintains information about which objects are being stored in which caches. In block 22, a writer 14 initiates a request to update an object. It contacts the consistency coordinator 12.

In block 23, the consistency coordinator 12 determines which caches, if any, are storing the object and for each cache including a copy of the object, the consistency coordinator 12 instructs the cache to delete its copy. After it receives acknowledgements that the deletions have completed, the consistency coordinator 12 informs the writer 14 that it can proceed with the update. If the object is frequently requested, in block 25, it may be

desirable to prefetch the object into one or more caches

after the update has completed. This step is optional.

There are a number of variations and options for the

coordinate-holders method. A method for coordinating

5   updates to an object when there are multiple writers is

described below. This method can be used in conjunction

with the coordinate-holders consistency scheme.

Also described herein is how cache failures can be handled

using heartbeats.

10   Deferred-invalidation consistency addresses the

problem of single-writer updates and provides strong

consistency in the case when the clocks of all nodes in the

system are perfectly synchronized. The method is based on

the idea that caches are instructed to discard the old

15   version of an object and start serving the most recent

version at a time in the future when each cache is likely

to have either learned about the update or declared itself

disconnected. The coordinator, based on the available

infrastructure mechanisms and configuration parameters, may

20   determine the length of this time interval.

The protocol is defined by the following steps. When the

content provider wishes to update an object it contacts the

consistency coordinator. The coordinator decides on the

time when the deferred invalidation has to be enacted by

the caches and sends to all caches a deferred-invalidation

message indicating the object and the time of invalidation.

Upon receiving this message, a cache marks the object for

5   invalidation at the indicated time (e.g., by setting the

expiration time to the indicated time), and sends an

acknowledgment to the coordinator.

Requests that are received by a cache between the

receipt of the deferred-invalidation message and the

10   invalidation time are replied with the old version of the

object. The first request after the invalidation time is

served the new version of the object. Caches that do not

acknowledge the deferred-invalidation message by the time

of the enactment are considered down by the coordinator.

15   Caches that have not received the deferred-invalidation

message are likely to have considered themselves down by

the time of the invalidation time, and caches that have

received the message but their acknowledgement does not

reach the coordinator, are likely to be either down or

20   enacting a correct invalidation at the invalidation time.

Multiple-writers Strong consistency addresses the

problem of multiple-writer updates in the context of

enforcing strong consistency among the caches storing the

object. In this method, before the update, the writer

contacts the consistency coordinator to retrieve the most

recent version of the object. The consistency coordinator

sends the content, or acknowledges that the local copy in

5   the writer cache is the most recent. Upon sending the

reply, the coordinator records a write lock for the object

held by the writer and assigns it a lock timeout.

Upon receiving the most recent version of the object,

the writer performs the update and sends the new version to

10   the consistency coordinator, which cancels the write lock,

and distributes the new content to the other caches using

either coordinate-all or coordinate-holders consistency

methods. To the writer cache, the coordinator sends an

acknowledgement of update upon receiving all of the

15   acknowledgements to the related invalidation requests. The

writer is not using the new version of the object to reply

to client requests until it receives an acknowledgement

from the coordinator. In the meantime, it can use the

previous version of the object to reply to requests that

20   only require a read of the updated object. If the writer

receives an invalidation request before the acknowledgment,

it discards both the old and the updated versions of the

object.

If the consistency coordinator receives another
request for update before the current write lock for the
object expires, it postpones the reply until the update is
received or the write lock expires. In the former case, the
new version is sent to the requesting node and a new write
lock is set for the object. In the latter case, the writer
cache is sent a negative acknowledgment of update, and the
requesting node is sent the version of the object available
to the coordinator and a new lock is set for the object.
Upon receiving a negative acknowledgement, the cache
invalidates the updated version, if already created, and it
can reinitiate the update procedure.

If an update completes before the previous version was
fully distributed to caches (according to the chosen
protocol), the coordinator saves the new content and acts
as indicated for update-local-copy if the second update
completes before the distribution completes.

Read requests arrived at the coordinator for an object
with a write lock are responded with the most recent
version available on the coordinator.

One issue of both Coordinate-all and Coordinate-
holders methods is that the caches may respond with very
different rates, some relatively fast while others

relatively slow. As a result, the updated object is not

accessible at faster responding caches for relatively long

time periods. During this period, pending requests from

clients are queued; thus, the response latency may be

5   unpredictably high.

Deferred-invalidation consistency addresses this drawback

by allowing the caches to serve the old version of the

update object until the system can guarantee that all of

the active caches are ready to serve the new version of the

10   object. Therefore, requests arrived at active caches will

never be blocked because other caches in the system fail to

respond to the update procedure. The drawback is that

updated content is available with a longer delay than for

Coordinate methods when all caches are active and fast

15   responding.

An issue with the Coordinate-all method is that on

each update, the consistency coordinator contacts each

cache in the configuration, whether or not the cache has a

copy of the updated object. This can result in unnecessary

20   network traffic if objects tend to be stored only in small

subsets of the caches.

The Coordinate-holders consistency addresses this

issue of the Coordinate-all consistency because only the

caches that have stored the object are involved in the consistency enforcement protocol. Deferred-invalidation consistency can be applied to coordinate all caches or only the holders of the updated object.

5   For Multiple-writers Strong consistency, the worst-case time of write completion includes a multiple of the write lock timeout and an invalidation timeout.

Cache Consistency Infrastructure

10   The present invention integrates the above consistency methods.

The system of the present invention includes at least one consistency coordinator 12 associated with the content provider server(s) 11 and several consistency slaves,

15   corresponding to remote caches 13, which store copies of objects produced by content providers and may update them as a result of client requests. The consistency slaves may be co-located with the corresponding caches and implement the cache counterpart of the consistency protocols.

20   The architecture of the present invention includes one or more consistency coordinators. Multiple consistency coordinators permit higher throughputs and higher availability. If one consistency coordinator fails, a

back-up consistency coordinator can take over for the

failed one. The functions performed by the coordinator may

include at least the following:

1. Maintain information about which caches are storing

5   which objects

2. Access and keep track of attributes of objects specified

   by the content provider. In particular, the coordinator

   should get the consistency policy to be used for an

   object.

10 3. Coordinate updates, through invalidation, to the caches

   upon request from content providers.

   Additionally, the coordinator can function as a

reverse proxy cache for the content provider, serving

requests for objects invalidated through consistency

15 protocols, and obviating the need for the content provider

to handle these requests.

   The coordinator handles several types of requests,

which may include the following:

   GET requests, which are used by caches to retrieve

20 objects of interest.

   IF-MOD-SINCE requests, which are used to check whether an

object was updated since a particular moment in the past,

and if so, to retrieve the new version of the object.

UPDATE requests, which are used by content

providers/writers to notify that a new version of an object

is available.

LOCK requests, which are used by content

5   providers/writers to notify their intent to initiate an

object update.

In the process of serving GET and IF-MOD-SINCE

requests the coordinator may retrieve the requested object

from the content provider, possibly saving it in a local

10   cache, and returning it to the requesting cache.

Alternatively, the coordinator may reply to the cache with

a REDIRECT message, indicating the node (cache or content

provider) to which the cache should send its request.

Both GET and IF-MOD-SINCE requests may be delayed when

15   the coordinator is in the process of updating the object.

The coordinator can implement a policy of choice for

handling requests received while the related object is

being updated. For example, the reply can be postponed

until all invalidations are complete, or an error message

20   can be sent immediately indicating the page is not

available.

An UPDATE request triggers the coordinator to begin the

consistency procedure. Based on the consistency policy of

the object, the coordinator sends invalidation messages to

caches and waits for acknowledgments from caches. For

objects with multiple writers/content providers, a writer

may issue a LOCK request prior to initiating the update

5   procedure.   Depending on the type of consistency of the

object, the writer may update its object-related information

to indicate that object is in process of being updated by

the writer. Also, the coordinator may delay the reply until

the UPDATE requests from writers previously locking the

10   object have been completed.

In the event of a failure, the coordinator may lose

part or all of its object and cache-related information. The

coordinator can use a number of techniques for reacquiring

information lost in the event of a failure. For example, the

15   coordinator may acquire, either immediately or over time,

information of which caches include which objects. One way

to do this is to demand immediately that all caches either

clear their caches or send to the coordinator the list of

the currently cached objects with update-holders and

20   coordinate-holders policies. Alternatively, the information

can be built up over time by invalidating caches for

objects, which have not been updated since the coordinator
has restarted.

The coordinator may be designed so that it can use a
variety of different protocols and mechanisms for
communicating with caches and servers. The coordinator can
also be adapted to perform functions not necessarily related
to consistency management, such as collecting statistical
information from the caches and monitoring
availability/responsiveness of the caches. If multiple
coordinators are being used, the coordinators can be
configured so that different coordinators manage different
subsets of the object space; possibly with the directory
hash partitioned among these components. This can provide
high scalability and availability.

Object meta information and state

An object usually has a consistency policy assigned to
it. For either of the strong consistency policies, an
object has two states, Serving and Updating. The Serving
state indicates that the object is consistent in all caches
and can be served by the coordinator. The Updating state
indicates that an update request for the object is in

process, and any request received for the object at the

coordinator should be queued until the update is completed

or replied to with an error message. This state begins

when the update request is received from the content

5   provider, and ends when all invalidation acknowledgements

have been received (or retried until timeout) and the new

version of the object can be made available.

      For either of the weak consistency policies, an object

usually has only one state, Serving, which indicates that

10   it can be served by the coordinator.

      A cache can be in one of three states:

      Available, which indicates that consistency-related

communication initiated by the coordinator with the cache

was completed correctly;

15   **Retry**, which indicates that the cache has not

responded to the most recent message sent by the

coordinator; and

      Down, which indicates that the cache is considered

failed.

20   The coordinator views a cache as Available, as long as

the cache is responding within a timeout period to the

messages sent by the coordinator. If the coordinator

experiences an error communicating with a cache, it changes

the state of the cache to Retry and continues to retry the

failed communication. If the communication succeeds within

an implementation-specific interval, the state of the cache

returns to Available. On the other hand, if the

communication fails, the cache is considered Down and no

further communication is sent to it until the cache sends a

"BackToLife" message, indicating that it would like to

recover its status since contact was lost. On receipt of

that request the coordinator and cache perform the

consistency recovery protocol.

To bound the latency of completing a strong

consistency protocol and the likelihood of inconsistency

for weak consistency protocols, the coordinator sends to

caches periodic heartbeat messages. Given the constant

stream of requests from the caches, the heartbeats need not

be in the form of separate messages; the presence of normal

message traffic could take its place except during idle

periods.

When a cache state is Available, heartbeat messages

are sent every heartbeat interval. In Retry state, a cache

is not sent heartbeats, but the coordinator is actively

retrying the failing communication for as long as a

heartbeat interval. If the message retry is successful,

normal heartbeat messages resume and no further action is required. If the heartbeat interval passes without an acknowledgment from the cache then the coordinator changes the state of the cache to Down. When the coordinator

5   changes the state to Down, the cache, if alive, declares itself Down as well, because it has not received any heartbeat message for the last heartbeat interval (because the server did not send any). In this state, the cache is not serving any object with coordinate-type or update-type

10   consistency policy, but it can serve objects with expiration-based consistency.

One aspect can be derived from noticing that the need to allow completion of the barrier synchronization during updates of strongly-consistent objects is different from

15   the need to keep caches from serving excessively stale weakly-consistent objects. These two needs may best be served by significantly different timeouts for the cache to use for passing from the Available state to the Down state with regard to strongly-consistent versus weakly-consistent

20   objects. For example, it may be felt that service of updates for strongly-consistent objects should never be delayed by more than 15 seconds, while it may be perfectly acceptable to allow service of weakly-consistent objects to

continue for up to 2 minutes after the update has taken

place.   Having separate timeout intervals for these 2 types

of objects would allow the lapse of service during update

of a strongly-consistent object to be kept to a reasonable

5   minimum while, at the same time, avoiding lapses in service

of weakly-consistent data due to unnecessarily stringent

timing demands on the caches' network connections to the

coordinator.

There are several types of requests or commands that

10   are received and sent by the coordinator in accordance with

the present invention. The coordinator's response depends

on the status of the cache and the status of the object.

The coordinator may also update its own status based on

receipt of the request. As a general procedure, when the

15   coordinator receives a command from a Down cache, other

than a request to recover, the coordinator returns an error

message that notifies the cache that it should be Down.

This causes the cache to perform recovery before it serves

more objects. This situation occurs when the coordinator

20   believes the cache has gone down but the cache does not

believe it is down.

GET request

The coordinator receives GET requests from a cache when it is asked to serve an object, which it is not in its cache, for example, a cache miss. The coordinator retrieves the requested object from the content provider (or from a local cache if appropriate) and returns it to the cache. When the object being requested has consistency policy of update-holders or coordinate-holders, a GET request indicates that the cache issuing the request now has this object in its cache and should be included in update processing. The coordinator updates its information to make note of this status change.

If the object is in state Updating (e.g., in the process of being updated with one of the coordinate-type policies), the GET request is queued until the update is complete or replied with an error message.

IF-MODIFIED-SINCE request

The coordinator receives IF-MODIFIED-SINCE requests when the cache includes an object, but may not contain the most recent version of the object. The coordinator processes the request as appropriate, returning a new version of the object if appropriate. When the object

being requested has consistency policy of update-holders or coordinate-holders, the coordinator updates its information appropriately.

5   If the object is in state Updating (e.g., in the process of being updated with one of the coordinate-type policies), the request is queued until the update is complete or replied to with an error message.

DISCARD request

10   The coordinator receives DISCARD requests when a cache chooses to discard an object that has update-holders or coordinate-holders policy. Upon receiving a DISCARD request, the coordinator updates its information to reflect that the cache is no longer storing the object.

UPDATE request

15   The coordinator receives an UPDATE request from a content provider or writer that notifies the coordinator that a new version of an object is available. The procedure executed upon receiving this command depends on the type of consistency of the updated object.

20

Weak consistency policies: update-all, update-holders,

update-local-copy

Upon receiving an update for an object with a weak

consistency policy, the coordinator refreshes the version

5   of the object, updating the meta-data information, and

possibly retrieving the new version of the object in the

local cache. The coordinator sends invalidate messages to

either all its associated caches, in the case of update-

all, or all caches known or suspected to have the object,

10   in the case of update-holders. The coordinator waits for

acknowledgments from the caches for the invalidate command,

and retries if necessary. If a cache fails to respond

after retrying for the heartbeat interval, the coordinator

declares that cache Down and stops communication with it

15   until that cache has performed recovery.

Weak consistency policies: Update-Global copy

Upon receiving an update for an object with update-

global copy consistency, the coordinator checks whether the

node is the current holder of the object lock. If this is

20   true, the indication that the node is the lock holder is

removed, and an update procedure described herein is

performed, and, eventually, the first node waiting in the

object's lock queue is granted the lock (e.g., sent a reply to its LOCK request). If the requesting node is not the lock holder, the update request is denied and the node is sent an error message.

5   Strong consistency policies: coordinate-all, coordinate-holders

Upon receiving an update for an object with a strong consistency policy, the coordinator updates the status of the object to Updating. This ensures that future requests

10   for the object are queued. Then, the coordinator sends invalidate messages to either all its associated caches, in the case of coordinate-all, or all caches known or suspected to have the object, in the case of coordinate-holders. The coordinator waits for acknowledgments from

15   caches for the invalidate command, and retries if needed. If a cache fails to respond after retrying for the heartbeat interval, the coordinator declares that cache Down and stops communication with it until that cache performs the recovery procedure. Once caches have

20   acknowledged the invalidate command or have been declared Down, the coordinator makes the new version of the object available and updates the object state to Available.

Deferred-Invalidation policy

Upon receiving an update for an object with a strong consistency policy, the coordinator determines the invalidation time and registers it in the object descriptor. Then, the coordinator sends deferred-invalidation messages to either all or the holder caches, depending on the configuration. The coordinator waits for acknowledgments from the caches for the invalidate command, and retries if needed. If a cache fails to respond after retrying for the heartbeat interval, the coordinator declares that cache Down and stops communication with it until that cache performs the recovery procedure. Requests that arrive at the coordinator prior to the invalidation time are served with the old version of the object. The first request received after the invalidation time triggers the actual update, by discarding the old version and retrieving the new version from the content provider or from the local repository.

Strong consistency policies: multiple-writers strong

Upon receiving an update for an object with update-global copy consistency, the coordinator checks whether the node is the current holder of the object lock. If this is

true, the indication that the node is the lock holder is
removed, an update procedure is performed, and, eventually,
the first node waiting in the object's lock queue is
granted the lock (e.g., sent a reply to its LOCK request).

5 If the requesting node is not the lock holder, the update
request is denied and the node is sent an error message.

LOCK request

The coordinator receives a LOCK request when a content
provider or writer decides to initiate an update procedure

10 for an object with multiple writers and consistency type
Update-Global Copy or Multiple-writers Strong. Upon
receiving the LOCK request, the coordinator checks whether
the object is being locked by another node. If this is
true, the requesting node is placed on the waiting queue of

15 the lock. If this is false, the object is marked as being
locked by the requesting node and the node is sent a reply
indicating the availability of the object for update and
the most recent version of the object. Optionally, the
reply may include the content of the most recent version of

20 the object.

CONSISTENCY-POLICY-CHANGE request

The coordinator receives a CONSISTENCY-POLICY-CHANGE request when a content provider notifies the coordinator when the consistency policy for the object has changed. If

5   a consistency policy change is received while an object is being updated, the currently active update is completed using the previous policy, and the new policy takes effect once the update is complete.

Changing to policy expiration-time, update-all, coordinate-

10   all

If the new policy is one, which does not need cache/object relationships to be maintained by the coordinator, then changing the policy of an object is relatively simple. Once active updates are complete the

15   coordinator removes state information about the object. This applies to changing to policies: expiration-time, update-all and coordinate-all.

Changing to policy update-holders or coordinate-holders

When changing to policy update-holders or coordinate-

20   holders the list of caches including the object should be built if the prior policy was update-all or coordinate-all.

In this case, the coordinator invalidates the object in caches. The function is similar to updating an object with policy update-all. Invalidations are sent to all caches and the coordinator waits for acknowledgments. Once all

5   caches acknowledge or are declared Down, the change is complete. During the period that the coordinator is waiting for acknowledgments no updates to the object are allowed, but GET requests are honored as if the new policy was in effect.

10   <u>Recover or BackToLife request</u>

Once a cache detects that it may have lost communication with the coordinator, normally via a missing heartbeat, it sends a Recover, or BackToLife, message to the coordinator. When the cache state at the coordinator

15   is Available, the coordinator response indicates that communication was not lost, meaning a heartbeat may have been lost but no updates happened during that time so that cache state is still valid. In this case no further processing is needed.

20   When the cache state is Down, the coordinator reply signals the cache to initialize the recovery procedure because the cache lost at least one invalidation message.

When the cache state is Retry, the coordinator reply
indicates that retry is taking place. Also, the
coordinator may extend the retry interval to ensure that
the retry will continue for at least a configuration-
specific constant. This helps minimize the likelihood of
declaring the cache down just after its connectivity
recovered, but it is a trade-off with the latency of a
strong consistency update.

Heartbeat notification

The coordinator sends heartbeat notifications to all
caches in state Available, at fixed time intervals. The
heartbeat interval is a system configuration parameter.
The cache does not have to acknowledge heartbeat messages,
but uses them to verify that the coordinator still
considers it alive. It is also possible within the spirit
and scope of the present invention to send heart beat
messages from a cache to the consistency coordinator.
Heartbeat messages do not have to be sent to a cache when
the coordinator is waiting for the cache to acknowledge a
command/message.

Invalidation notification

The coordinator sends Invalidation notifications to one or more caches in state Available to indicate that particular objects should be discarded from their local

5   stores. These messages are triggered by UPDATE requests. Depending on the type of consistency of the invalidated objects, caches may have to acknowledge the receipt of an Invalidation notification.

Consistency Slave

10   The consistency slave is a module loaded on the cache node. The functions of this module may include the following:

1. track of consistency state of the various objects in the local cache; and

15   2. interact with consistency coordinator.

The consistency slave configuration parameters include the address of consistency coordinator(s). In systems with multiple consistency coordinators, it is assumed that the mapping of objects to consistency coordinators is defined

20   by configuration parameters.

Data structures for the consistency slave will now be described. The consistency slave maintains state for the

objects with coordinate-holders and update-holders

consistency policies. The presence of an object ID on a

list maintained by a consistency slave indicates that the

cache has to send a discard request when the object is

5   removed from the cache. The Consistency Slave maintains

state for the objects currently locked by the cache

applications. Also, the consistency slave maintains state

regarding the connectivity of the local node to the rest of

the system, in particular to the consistency coordinator.

10 The per-object state of the consistency slave may be

maintained separately or may be integrated with the state

maintained by the cache application.

The cache application invokes the consistency *slave*

when it needs to read or write an object, and when it

15 discards an object from its local store.

Read Command

The Read command is invoked when the cache has to

serve a read request. The call parameters provide the

object identifier, and metadata information such as the

20 existence of the object in the cache. If the object is

registered with the consistency slave and the metadata

indicates a consistency type that does not need consistency

checks, the call returns with the indication that the cache

application should handle the object itself. Otherwise, if

the consistency slave knows the consistency type of the

object, it executes the specific consistency protocol.   If

5   the consistency type is not known yet (e.g., when object is

not in local cache), the slave interacts with the

consistency coordinator to retrieve the object's

characteristics and, optionally, the associated content.

Eventually, the slave returns to the cache application with

10   an indication of whether a local copy is valid or the cache

should retrieve the object from an indicated location.

Read-for-Update command

This command is invoked by the cache application when

15   it has to initiate an update operation. The call

parameters provide the object identifier, and metadata

information such as the existence of the object in the

cache. If the object is registered with the consistency

slave and the metadata indicates a consistency type that

20   does not need any consistency-related procedure, the call

returns with the indication that the cache application

should handle the object itself. Otherwise, if the

consistency slave knows the consistency type of the object,
it executes the specific consistency protocol. For
instance, if the policy is Update-Global Copy, the slave
interacts with the coordinator to acquire the lock on the
5   object. If the consistency type is not known yet (e.g.,
when object is not in local cache), the slave interacts
with the consistency coordinator to retrieve the object's
characteristics and, optionally, the associated content.
Eventually, the slave returns to the cache application with
10   an indication of whether a local copy is valid or the cache
should retrieve the object from an indicated location, and
on whether the cache should create the new version of the
object without overriding the current version.

Update-Completion command

15   This command is invoked by the cache application when
it completes an update operation. The call parameters
provide the object identifier, indication of whether the
update completes successfully or it was aborted, and the
location of the new version (if successful update).
20   Depending on the consistency type of the object, the
consistency slave interacts with the coordinator to
indicate the completion of the operation.

Discard command

This command is invoked by the cache application when it discards an object from the local store. The consistency slave executes the protocol specific for the object type. No specific information is returned to the cache application.

The consistency slave learns about the type of consistency associated with an object from the metadata attached to the replies to its GET and LOCK requests to the consistency coordinator.
Object invalidations and acknowledgements, (deferred) removal notifications, and heartbeat messages may be delivered through messages on a persistent connection between the cache node and consistency coordinator node.

The interaction between the slave and the coordinator can be embedded in HTTP messages or they can be implemented by other protocols. In the former case, GET, IF MODIFIED SINCE, and LOCK requests can be sent with HTTP GET requests. UPDATE, CONSISTENCY-POLICY-CHANGE, and RECOVER requests can be sent with HTTP POST requests. Similarly, INVALIDATION and HEARTBEAT messages can be sent with HTTP

POST requests. The messages initiated by the coordinator,

such as HEARTBEAT and INVALIDATION messages, are received

at a designated port of the cache node, which can be

handled by the consistency slave module itself or by cache

5   application. In the former case, the consistency slave

interface includes a callback function, which is invoked by

the cache application upon arrival of a message on the

designed port.

10   Batch removal notifications

        For the update-holders and coordinate-holders

policies, the slaves send notifications of cache removal

when objects are discarded from their caches. To reduce

the overhead, these notifications can be batched in

15   messages of up to MAX-MSG-SIZE bytes. These messages are

sent when the maximum size is reached or a predefined time

interval has elapsed since the first notification in the

message was generated.

        Due to batching or network delays, the coordinator can

20   receive removal and get requests in reverse logical order,

e.g., the GET following a removal GET arrive at the

coordinator a priori to the removal notification. To

ensure a correct accounting, the coordinator keeps track of

the number of requests and removals received for a

particular (object, cache)-pair for objects subject to

update-holders or coordinate-holders policy. On each

request, the counter is incremented, and on each removal

5   the counter is decremented. The server removes the cache

from the holders list for the object when the counter gets

to zero.

Aggregation of Consistency Protocol Messages

To reduce the overhead related to the transmission of

10   consistency protocol messages, consistency coordinators

and/or or consistency slaves can aggregate several messages

in one packet. For instance, Invalidation messages sent by

the consistency coordinator can include the ID's of several

objects. Similarly, the Acknowledgment message sent by a

15   cache can include the ID's of several objects.

For further overhead reductions, the consistency

infrastructure enables the specification of consistency

groups. Toward this end, an object is identified by the

content provider by its ID and the list of consistency

20   groups it belongs to. Update requests for a consistency

group should trigger the invalidation of all of the objects

in the group.

In this way, it is not necessary to enumerate each object in the group explicitly. Data update propagation (see e.g., "A Scalable System for Consistently Caching Dynamic Web Data", Jim Challenger, Arun Iyengar,

5   and Paul Dantzig. In *Proceedings of IEEE INFOCOM'99*, New York, New York, March 1999) may be used to specify group membership.

Prefetch/Push for Deferred Consistency Protocol

Servers and/or content providers may have the ability

10   to prefetch or push a new version of an object to a cache.

Having described preferred embodiments of a system and method for achieving strong data consistency (which are intended to be illustrative and not limiting), it is noted that modifications and variations can be made by persons

15   skilled in the art in light of the above teachings. It is therefore to be understood that changes may be made in the particular embodiments of the invention disclosed which are within the scope and spirit of the invention as outlined by the appended claims. Having thus described the invention

20   with the details and particularity required by the patent laws, what is claimed and desired protected by Letters Patent is set forth in the appended claims.